File System Checking

This document is part of the ADMINISTRATOR'S GUIDE. Therefore the pagenumbers don't begin with 1.

8. FILE SYSTEM CHECKING

The File System Check Program (**fsck**) is an interactive file system check and repair program. **Fsck** uses the redundant structural information in the UNIX file system to perform several consistency checks. If an inconsistency is detected, it is reported to the operator, who may elect to fix or ignore each inconsistency. These inconsistencies result from the permanent interruption of the file system updates, which are performed every time a file is modified. **Fsck** is frequently able to repair corrupted file systems using procedures based upon the order in which the UNIX system honors these file system update requests.

The purpose of this section is to describe the normal updating of the file system, to discuss the possible causes of file system corruption, and to present the corrective actions implemented by **fsck**. Both the program and the interaction between the program and the operator are described.

Appendix 8.1 contains the **fsck** error conditions. The meaning of the various error conditions, possible responses, and related error conditions are explained.

GENERAL

When a UNIX operating system is brought up, a consistency check of the file systems should always be performed. This precautionary measure helps to ensure a reliable environment for file storage on disk. If an inconsistency is discovered, corrective action must be taken.

The purpose of this section is to dispel the mystique surrounding file system inconsistencies. The section describes the 5.0 file system, the updating of the file system, and then describes file system corruption. Finally, the set of heuristically sound corrective actions used by **fsck** are presented.

THE 5.0 FILE SYSTEM

A. Introduction

The 5.0 file system features a larger internal block size. The block size increased from 512 bytes to 1024 bytes and increased the performance of I/O bound applications. The size of the internal system buffers also increased to 1024 bytes. For a 1024-byte block file system, data transfers to/from disk are in 1024-byte operations.

B. Description

A 512-byte block file system is still supported by the operating system and file system related commands. Both file system sizes are allowed to coexist by detecting the file system type as set in the superblock. At file system mounting time, the operating system checks the magic number and type fields in the superblock. This magic number is unique in the sense that it is unlikely to be matched by an old 512-byte file system. A magic number mismatch assumes an original 512-byte block. New 1024-byte block file systems should have the special magic number set in the superblock and type field specifying a 1024-byte block. These fields are set at file system creation time (*/etc/mkfs*). Also, new file systems with 512-byte blocks may be created. These will have the special magic number set and type field indicating a 512-byte block. These fields in the superblock are set at creation time (*/etc/omkfs*). Labelit will report the file system type.

No functional changes should be perceived by the user. File system related commands have changed internally to handle both types of file systems. These changes are transparent to the user; the user interface remains unchanged. Most commands still report in 512-byte block units.

The root file system will be distributed as a 1024-byte block file system. Users are encouraged to convert their old file systems to the larger size block. However, 512-byte block file systems are still acceptable.

C. System Administrator Advice

Remember that system buffers are now 1024 bytes. When configuring the operating system, take into consideration that the same number of buffers as before will use more main memory. Weigh this against reducing the number of buffers, which reduces the cache hit ratio and degrades performance.

UPDATE OF THE FILE SYSTEM

Every working day hundreds of files are created, modified, and removed. Every time a file is modified, the UNIX operating system performs a series of file system updates. These updates, when written on disk, yield a consistent file system. To understand what happens in the event of a permanent interruption in this sequence, it is important to understand the order in which the update requests were probably being honored. Knowing which pieces of information were probably written to the file system first, heuristic procedures can be developed to repair a corrupted file system.

There are five types of file system updates. These involve the superblock, inodes, indirect blocks, data blocks (directories and files), and free-list blocks.

A. Superblock

The superblock contains information about the size of the file system, the size of the inode list, part of the free-block list, the count of free blocks, the count of free inodes, and part of the free-inode list.

The superblock of a mounted file system (the root file system is always mounted) is written to the file system whenever the file system is unmounted or a **sync** command is issued.

B. Inodes

An inode contains information about the type of inode (directory, data, or special), the number of directory entries linked to the inode, the list of blocks claimed by the inode, and the size of the inode.

An inode is written to the file system upon closure of the file associated with the inode. (All "in" core blocks are also written to the file system upon issue of a **sync** system call.)

C. Indirect Blocks

There are three types of indirect blocks—single indirect, double indirect, and triple indirect. A single-indirect block contains a list of some of the block numbers claimed by an inode. Each one of the 128 entries in an indirect block is a data-block number. A double-indirect block contains a list of single-indirect block numbers. A triple-indirect block contains a list of double-indirect block numbers.

Indirect blocks are written to the file system whenever they have been modified and released by the operating system. More precisely, they are queued for eventual writing. Physical I/O is deferred until the buffer is needed by the UNIX system or a **sync** command is issued.

D. Data Blocks

A data block may contain file information or directory entries. Each directory entry consists of a file name and an inode number.

Data blocks are written to the file system whenever they have been modified and released by the operating system.

E. First Free-List Block

The superblock contains the first free-list block. The free-list blocks are a list of all blocks that are not allocated to the superblock, inodes, indirect blocks, or data blocks. Each free-list block contains a count of the number of entries in this free-list block, a pointer to the next free-list block, and a partial list of free blocks in the file system.

Free-list blocks are written to the file system whenever they have been modified and released by the operating system.

CORRUPTION OF THE FILE SYSTEM

A file system can become corrupted in a variety of ways. The most common of these ways are improper shutdown procedures and hardware failures.

A. Improper System Shutdown and Startup

File systems may become corrupted when proper shutdown procedures are not observed, e.g., forgetting to **sync** the system prior to halting the CPU, physically write-protecting a mounted file system, or taking a mounted file system off-line.

File systems may become further corrupted if proper startup procedures are not observed, e.g., not checking a file system for inconsistencies and not repairing inconsistencies. Allowing a corrupted file system to be used (and, thus, to be modified further) can be disastrous.

B. Hardware Failure

Any piece of hardware can fail at any time. Failures can be as subtle as a bad block on a disk pack or as blatant as a nonfunctional disk-controller.

DETECTION AND CORRECTION OF CORRUPTION

A quiescent file system (i.e., an unmounted system and not being written on) may be checked for structural integrity by performing consistency checks on the redundant data intrinsic to a file system. The redundant data is either read from the file system or computed from other known values. A quiescent state is important during the checking of a file system because of the multipass nature of the **fsck** program.

When an inconsistency is discovered, **fsck** reports the inconsistency for the operator to chose a corrective action.

Discussed in this part are how to discover inconsistencies (and possible corrective actions) for the superblock, the inodes, the indirect blocks, the data blocks containing directory entries, and the free-list blocks. These corrective actions can be performed interactively by the **fsck** command under control of the operator.

A. Superblock

One of the most common corrupted items is the superblock. The superblock is prone to corruption because every change to the file system's blocks or inodes modifies the superblock.

The superblock and its associated parts are most often corrupted when the computer is halted and the last command involving output to the file system was not a **sync** command.

The superblock can be checked for inconsistencies involving file-system size, inode-list size, free-block list, free-block count, and the free-inode count.

File-System Size and Inode-List Size

The file-system size must be larger than the number of blocks used by the superblock and the number of blocks used by the list of inodes. The number of inodes must be less than 65,535. The file-system size and inode-list size are critical pieces of information to the **fsck** program. While there is no way to actually check these sizes, **fsck** can check for them being within reasonable bounds. All other checks of the file system depend on the correctness of these sizes.

Free-Block List

The free-block list starts in the superblock and continues through the free-list blocks of the file system. Each free-list block can be checked for a list count out of range, for block numbers out of range, and for blocks already allocated within the file system. A check is made to see that all the blocks in the file system were found.

The first free-block list is in the superblock. **Fsck** checks the list count for a value of less than 0 or greater than 50. It also checks each block number for a value of less than the first data block in the file system or greater than the last block in the file system. Then it compares each block number to a list of already allocated blocks. If the free-list block pointer is nonzero, the next free-list block is read in and the process is repeated.

When all the blocks have been accounted for, a check is made to see if the number of blocks used by the free-block list plus the number of blocks claimed by the inodes equals the total number of blocks in the file system.

If anything is wrong with the free-block list, then **fsck** may rebuild the list, excluding all blocks in the list of allocated blocks.

Free-Block Count

The superblock contains a count of the total number of free blocks within the file system. **Fsck** compares this count to the number of blocks it found free within the file system. If the counts do not agree, then **fsck** may replace the count in the superblock by the actual free-block count.

Free-Inode Count

The superblock contains a count of the total number of free inodes within the file system. **Fsck** compares this count to the number of inodes it found free within the file system. If the counts do not agree, then **fsck** may replace the count in the superblock by the actual free-inode count.

B. Inodes

An individual inode is not as likely to be corrupted as the superblock. However, because of the great number of active inodes, there is almost as likely a chance for corruption in the inode list as in the superblock.

The list of inodes is checked sequentially starting with inode 1 (there is no inode 0) and going to the last inode in the file system. Each inode can be checked for inconsistencies involving format and type, link count, duplicate blocks, bad blocks, and inode size.

Format and Type.

Each inode contains a mode word. This mode word describes the type and state of the inode. Inodes may be one of four types:

- regular
- directory

- special block
- special character.

If an inode is not one of these types, then the inode has an illegal type. Inodes may be found in one of three states—unallocated, allocated, and neither unallocated nor allocated. This last state indicates an incorrectly formatted inode. An inode can get in this state if bad data is written into the inode list through, for example, a hardware failure. The only possible corrective action is for **fsck** to clear the inode.

Link Count

Contained in each inode is a count of the total number of directory entries linked to the inode. **Fsck** verifies the link count of each inode by traversing down the total directory structure, starting from the root directory, and calculating an actual link count for each inode.

If the stored link count is nonzero and the actual link count is zero, it means that no directory entry appears for the inode. If the stored and actual link counts are nonzero and unequal, a directory entry may have been added or removed without the inode being updated.

If the stored link count is nonzero and the actual link count is zero, **fsck** may link the disconnected file to the *lost+found* directory. If the stored and actual link counts are nonzero and unequal, **fsck** may replace the stored link count by the actual link count.

Duplicate Blocks

Contained in each inode is a list or pointers to lists (indirect blocks) of all the blocks claimed by the inode. **Fsck** compares each block number claimed by an inode to a list of already allocated blocks. If a block number is already claimed by another inode, the block number is added to a list of duplicate blocks. Otherwise, the list of allocated blocks is updated to include the block number. If there are any duplicate blocks, **fsck** will make a partial second pass of the inode list to find the inode of the duplicated block. This is necessary because without examining the files associated with these inodes for correct content there is not enough information available to decide which inode is corrupted and should be cleared. Most times, the inode with the earliest modify time is incorrect and should be cleared. This condition can occur by using a file system with blocks claimed by both the free-block list and by other parts of the file system.

If there is a large number of duplicate blocks in an inode, this may be due to an indirect block not being written to the file system. **Fsck** will prompt the operator to clear both inodes.

Bad Blocks

Contained in each inode is a list or pointer to lists of all the blocks claimed by the inode. **Fsck** checks each block number claimed by an inode for a value lower than that of the first data block or greater than the last block in the file system. If the block number is outside this range, the block number is a bad block number.

If there is a large number of bad blocks in an inode, this may be due to an indirect block not being written to the file system. **Fsck** will prompt the operator to clear both inodes.

Size Checks

Each inode contains a 32 bit (4-byte) size field. This size indicates the number of characters in the file associated with the inode. This size can be checked for inconsistencies, e.g., directory sizes that are not a multiple of 16 characters or the number of blocks actually used not matching that indicated by the inode size.

A directory inode within the UNIX file system has the directory bit on in the inode mode word. The directory size must be a multiple of 16 because a directory entry contains 16 bytes (2 bytes for the inode number and 14 bytes for the file or directory name).

Fsck will warn of such directory misalignment. This is only a warning because not enough information can be gathered to correct the misalignment.

A rough check of the consistency of the size field of an inode can be performed by computing from the size field the number of blocks that should be associated with the inode and comparing it to the actual number of blocks claimed by the inode.

Fsck calculates the number of blocks that there should be in an inode by dividing the number of characters in an inode by the number of characters per block and rounding up. **Fsck** adds one block for each indirect block associated with the inode. If the actual number of blocks does not match the computed number of blocks, **fsck** will warn of a possible file-size error. This is only a warning because the UNIX system does not fill in blocks in files created in random order.

C. Indirect Blocks

Indirect blocks are owned by an inode. Therefore, inconsistencies in indirect blocks directly affect the inode that owns it.

Inconsistencies that can be checked are blocks already claimed by another inode and block numbers outside the range of the file system.

For a discussion of detection and correction of the inconsistencies associated with indirect blocks, see the parts "Duplicate Blocks" and "Bad Blocks".

D. Data Blocks

The two types of data blocks are plain data blocks and directory data blocks. Plain data blocks contain the information stored in a file. Directory data blocks contain directory entries. **Fsck** does not attempt to check the validity of the contents of a plain data block.

Each directory data block can be checked for inconsistencies involving directory inode numbers pointing to unallocated inodes, directory inode numbers greater than the number of inodes in the file system, incorrect directory inode numbers for "." and "..", and directories which are disconnected from the file system. In addition, the validity of the contents of a directory's data block is checked.

If a directory entry inode number points to an unallocated inode, then **fsck** may remove that directory entry. This condition probably occurred because the data blocks containing the directory entries were modified and written out while the inode was not yet written out.

If a directory entry inode number is pointing beyond the end of the inode list, **fsck** may remove that directory entry. This condition occurs if bad data is written into a directory data block.

The directory inode number entry for "." should be the first entry in the directory data block. Its value should be equal to the inode number for the directory data block.

The directory inode number entry for ".." should be the second entry in the directory data block. Its value should be equal to the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory).

If the directory inode numbers are incorrect, **fsck** may replace them by the correct values.

Fsck checks the general connectivity of the file system. If directories are found not to be linked into the file system, **fsck** will link the directory back into the file system in the *lost+found* directory. This condition

can be caused by inodes being written to the file system with the corresponding directory data blocks not being written to the file system.

E. Free-List Blocks

Free-list blocks are owned by the superblock. Therefore, inconsistencies in free-list blocks directly affect the superblock.

Inconsistencies that can be checked are a list count outside of range, block numbers outside of range, and blocks already associated with the file system.

For a discussion of detection and correction of the inconsistencies associated with free-list blocks, see the part "Free-Block List".

APPENDIX 8.1

FSCK ERROR CONDITIONS

A. Conventions

Fsck is a multipass file system check program. Each file system pass invokes a different phase of the **fsck** program. After the initial setup, **fsck** performs successive phases over each file system performing cleanup, checking blocks and sizes, pathnames, connectivity, reference counts, and the free-block list (possibly rebuilding it).

When an inconsistency is detected, **fsck** reports the error condition to the operator. If a response is required, **fsck** prints a prompt message and waits for a response. This appendix explains the meaning of each error condition, the possible responses, and the related error conditions.

The error conditions are organized by the "Phase" of the **fsck** program in which they can occur. The error conditions that may occur in more than one phase will be discussed under the part "Initialization".

B. Initialization

Before a file system check can be performed, certain tables have to be set up and certain files opened. This part concerns itself with the opening of files and the initialization of tables. Error conditions resulting from command line options, memory requests, opening of files, status of files, file system size checks, and creation of the scratch file are listed below.

C option?

C is not a legal option to **fsck**; legal options are **−y**, **−n**, **−s**, **−S**, **−t**, **−f**, **−q**, and **−D**. **Fsck** terminates on this error condition. See the **fsck**(1M) entry in the UNIX System Administrator's Manual for further details.

Bad −t option

The **−t** option is not followed by a file name. **Fsck** terminates on this error condition. See the **fsck**(1M) entry in the UNIX System Administrator's Manual for further details.

Invalid −s argument, defaults assumed

The **−s** option is not suffixed by 3, 4, or blocks-per-cylinder:blocks-to-skip. **Fsck** assumes a default value of 400 blocks-per-cylinder and 7 blocks-to-skip. See the **fsck**(1M) entry in the UNIX System Administrator's Manual for further details.

Incompatible options: −n and −s

It is not possible to salvage the free-block list without modifying the file system. **Fsck** terminates on this error condition. See the **fsck**(1M) entry in the UNIX System Administrator's Manual for further details.

Incompatible options: −n and −q

It is not possible to do automatic removal without modifying the file system. **Fsck** terminates on this error condition. See the **fsck**(1M) entry in the UNIX System Administrator's Manual for further details.

Can not fstat standard input

Fsck's attempt to **fstat** standard input failed. This should never happen. **Fsck** terminates on this error condition.

Can not get memory

Fsck's request for memory for its virtual memory tables failed. This should never happen. **Fsck** terminates on this error condition.

Can not open checklist file: F

The default file system checklist file *F* (usually */etc/checklist*) can not be opened for reading. **Fsck** terminates on this error condition. Check access modes of *F*.

Can not stat root

Fsck's request for statistics about the root directory "/" failed. This should never happen. **Fsck** terminates on this error condition.

Can not stat F

Fsck's request for statistics about the file system *F* failed. It ignores this file system and continues checking the next file system given. Check access modes of *F*.

FS is a mounted file system, ignored

This is to avoid modifying a mounted file system. It ignores this file system and continues with the next file system given.

F is not a block or character device

Fsck has been given a regular file name by mistake. It ignores this file system and continues checking the next file system given. Check file type of *F*.

Can not open F

The file system *F* can not be opened for reading. It ignores this file system and continues checking the next file system given. Check access modes of *F*.

Size check: fsize X isize Y

More blocks are used for the inode list *Y* than there are blocks in the file system *X*, or there are more than 65,535 inodes in the file system. It ignores this file system and continues checking the next file system given.

Can not create F

Fsck's request to create a scratch file *F* failed. It ignores this file system and continues checking the next file system given. Check access modes of *F*.

CAN NOT SEEK: BLK B (CONTINUE)

Fsck's request for moving to a specified block number *B* in the file system failed. This should never happen.

Possible responses to CONTINUE prompt are:

YES — Attempt to continue to run file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of

fsck should be made to recheck this file system. If block was part of the virtual memory buffer cache, **fsck** will terminate with the message "Fatal I/O error".

NO — Terminate program.

CAN NOT READ: BLK B (CONTINUE)

Fsck's request for reading a specified block number *B* in the file system failed. This should never happen.

Possible responses to CONTINUE prompt are:

YES — Attempt to continue to run file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system. If block was part of the virtual memory buffer cache, **fsck** will terminate with the message "Fatal I/O error".

NO — Terminate program.

CAN NOT WRITE: BLK B (CONTINUE)

Fsck's request for writing a specified block number *B* in the file system failed. The disk is write-protected.

Possible responses to CONTINUE prompt are:

YES — Attempt to continue to run file system check. Often, however, the problem will persist. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system. If block was part of the virtual memory buffer cache, **fsck** will terminate with the message "Fatal I/O error".

NO — Terminate program.

C. PHASE 1: CHECK BLOCKS AND SIZES

This phase concerns itself with the inode list. This part lists error conditions resulting from checking inode types, setting up the zero-link-count table, examining inode block numbers for bad or duplicate blocks, checking inode size, and checking inode format.

UNKNOWN FILE TYPE I=I (CLEAR)

The mode word of the inode *I* indicates that the inode is not a special character inode, special character inode, regular inode, or directory inode. See the part "Format and Types" for more information.

Possible responses to CLEAR prompt are:

YES — Deallocate inode *I* by zeroing its contents. This will always invoke the UNALLOCATED error condition in Phase 2 for each directory entry pointing to this inode.

NO — Ignore this error condition.

LINK COUNT TABLE OVERFLOW (CONTINUE)

An internal table for **fsck** containing allocated inodes with a link count of zero has no more room. Recompile **fsck** with a larger value of MAXLNCNT.

Possible responses to CONTINUE prompt are:

YES — Continue with program. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system. If another allocated inode with a zero link count is found, this error condition is repeated.

NO — Terminate program.

B BAD I=I

Inode *I* contains block number *B* with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system. This error condition may invoke the EXCESSIVE BAD BLKS error condition in Phase 1 if inode *I* has too many block numbers outside the file system range. This error condition will always invoke the BAD/DUP error condition in Phase 2 and Phase 4. See the part "Bad Blocks" for more information.

EXCESSIVE BAD BLKS I=I (CONTINUE)

There is more than a tolerable number (usually 10) of blocks with a number lower than the number of the first data block in the file system or greater than the number of last block in the file system associated with inode *I*. See the part "Bad Blocks" for more information.

Possible responses to CONTINUE prompt are:

YES — Ignore the rest of the blocks in this inode and continue checking with next inode in the file system. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system.

NO — Terminate program.

B DUP I=I

Inode *I* contains block number *B* which is already claimed by another inode. This error condition may invoke the EXCESSIVE DUP BLKS error condition in Phase 1 if inode *I* has too many block numbers claimed by other inodes. This error condition will always invoke Phase 1b and the BAD/DUP error condition in Phase 2 and Phase 4. See the part "Duplicate Blocks" for more information.

EXCESSIVE DUP BLKS I=I (CONTINUE)

There is more than a tolerable number (usually 10) of blocks claimed by other inodes. See the part "Duplicate Blocks" for more information.

Possible responses to CONTINUE prompt are:

YES — Ignore the rest of the blocks in this inode and continue checking with next inode in the file system. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system.

NO — Terminate program.

DUP TABLE OVERFLOW (CONTINUE)

An internal table in **fsck** containing duplicate block numbers has no more room. Recompile **fsck** with a larger value of DUPTBLSIZE.

Possible responses to CONTINUE prompt are:

YES — Continue with program. This error condition will not allow a complete check of the file system. A second run of **fsck** should be made to recheck this file system. If another duplicate block is found, this error condition will repeat.

NO — Terminate program.

POSSIBLE FILE SIZE ERROR I=I

The inode *I* size does not match the actual number of blocks used by the inode. This is only a warning. (See the part "Size Checks".) If the **−q** option is used, this message is not printed.

DIRECTORY MISALIGNED I=I

The size of a directory inode is not a multiple of the size of a directory entry (usually 16). This is only a warning. (See the part "Size Checks".) If the **−q** option is used, this message is not printed.

PARTIALLY ALLOCATED INODE I=I (CLEAR)

Inode *I* is neither allocated nor unallocated. See the part "Format and Types" for more information.

Possible responses to CLEAR prompt are:

YES — Deallocate inode **I** by zeroing its contents.

NO — Ignore this error condition.

D. PHASE 1B: RESCAN FOR MORE DUPS

When a duplicate block is found in the file system, the file system is rescanned to find the inode which previously claimed that block. This part lists the error condition when the duplicate block is found.

B DUP I=I

Inode *I* contains block number *B* which is already claimed by another inode. This error condition will always invoke the BAD/DUP error condition in Phase 2. Inodes with overlapping blocks may be determined by examining this error condition and the DUP error condition in Phase 1. See the part "Duplicate Blocks" for more information.

E. PHASE 2: CHECK PATHNAMES

This phase concerns itself with removing directory entries pointing to error conditioned inodes from Phase 1 and Phase 1b. This part lists error conditions resulting from root inode mode and status, directory inode pointers in range, and directory entries pointing to bad inodes.

ROOT INODE UNALLOCATED. TERMINATING

The root inode (always inode number 2) has no allocate mode bits. This should never happen. The program will terminate. See the part "Format and Types" for more information.

ROOT INODE NOT DIRECTORY (FIX)

The root inode (usually inode number 2) is not directory inode type.

Possible responses to FIX prompt are:

YES — Replace the root inode's type to be a directory. If the root inode's data blocks are not directory blocks, a *very* large number of error conditions will be produced.

NO — Terminate program.

DUPS/BAD IN ROOT INODE (CONTINUE)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks in the root inode (usually inode number 2) for the file system.

Possible responses to CONTINUE prompt are:

YES — Ignore DUPS/BAD error condition in root inode and attempt to continue to run the file system check. If root inode is not correct, then this may result in a large number of other error conditions.

NO — Terminate program.

I OUT OF RANGE I=I NAME=F (REMOVE)

A directory entry *F* has an inode number *I* which is greater than the end of the inode list. See the part "Data Blocks" for more information.

Possible responses to REMOVE prompt are:

YES — The directory entry *F* is removed.

NO — Ignore this error condition.

UNALLOCATED I=I OWNER=O MODE=M SIZE=S MTIME=T NAME=F (REMOVE)

A directory entry *F* has an inode *I* without allocate mode bits. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed. If the file system is not mounted and the **−n** option was not specified, the entry will be removed automatically if the inode it points to is character size 0.

Possible responses to REMOVE prompt are:

YES — The directory entry *F* is removed.

NO — Ignore this error condition.

DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T DIR=F (REMOVE)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, directory inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and directory name *F* are printed.

Possible responses to REMOVE prompt are:

YES — The directory entry *F* is removed.

NO — Ignore this error condition.

DUP/BAD I=I OWNER=O MODE=M SIZE=S MTIME=T FILE=F (REMOVE)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed.

Possible responses to REMOVE prompt are:

YES — The directory entry *F* is removed.

NO — Ignore this error condition.

BAD BLK B IN DIR I=I OWNER=O MODE=M SIZE=S MTIME=T

A bad block was found in DIR inode *I*. This message only occurs when the **−q** option is used. Error conditions looked for in directory blocks are nonzero padded entries, inconsistent "." and ".." entries, and imbedded slashes in the name field. This error message indicates that the user should at a later time either remove the directory inode if the entire block looks bad or change (or remove) those directory entries that look bad.

F. PHASE 3: CHECK CONNECTIVITY

This phase concerns itself with the directory connectivity seen in Phase 2. This part lists error conditions resulting from unreferenced directories and missing or full *lost+found* directories.

UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)

The directory inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. **Fsck** will force the reconnection of a nonempty directory.

Possible responses to RECONNECT prompt are:

YES — Reconnect directory inode *I* to the file system in directory for lost files (usually *lost+found*). This may invoke *lost+found* error condition in Phase 3 if there are problems connecting directory inode *I* to *lost+found*. This may also invoke CONNECTED error condition in Phase 3 if link was successful.

NO — Ignore this error condition. This will always invoke UNREF error condition in Phase 4.

SORRY. NO lost+found DIRECTORY

There is no *lost+found* directory in the root directory of the file system; **fsck** ignores the request to link a directory in *lost+found*. This will always invoke the UNREF error condition in Phase 4. Check access modes of *lost+found*. See **fsck**(1M) in the UNIX System Administrator's Manual for further details.

SORRY. NO SPACE IN lost+found DIRECTORY

There is no space to add another entry to the *lost+found* directory in the root directory of the file system; **fsck** ignores the request to link a directory in *lost+found*. This will always invoke the UNREF error condition in Phase 4. Clean out unnecessary entries in *lost+found* or make *lost+found* larger. See **fsck**(1M) in the UNIX System Administrator's Manual for further details.

DIR I=I1 CONNECTED. PARENT WAS I=I2

This is an advisory message indicating a directory inode *I1* was successfully connected to the *lost+found* directory. The parent inode *I2* of the directory inode *I1* is replaced by the inode number of the *lost+found* directory. See the parts "Link Count" and "Data Blocks" for more information.

G. PHASE 4: CHECK REFERENCE COUNTS

This phase concerns itself with the link count information seen in Phase 2 and Phase 3. This part lists error conditions resulting from unreferenced files, missing or full *lost+found* directory, incorrect link counts for files,

directories, or special files, unreferenced files and directories, bad and duplicate blocks in files and directories, and incorrect total free-inode counts.

UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (RECONNECT)

Inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. (See the part "Link Count".) If the **−n** option is not set and the file system is not mounted, empty files will not be reconnected and will be cleared automatically.

Possible responses to RECONNECT prompt are:

YES — Reconnect inode *I* to file system in the directory for lost files (usually *lost+found*). This may invoke *lost+found* error condition in Phase 4 if there are problems connecting inode *I* to *lost+found*.

NO — Ignore this error condition. This will always invoke CLEAR error condition in Phase 4.

SORRY. NO lost+found DIRECTORY

There is no *lost+found* directory in the root directory of the file system; **fsck** ignores the request to link a file in *lost+found*. This will always invoke CLEAR error condition in Phase 4. Check access modes of *lost+found*.

SORRY. NO SPACE IN lost+found DIRECTORY

There is no space to add another entry to the *lost+found* directory in the root directory of the file system; **fsck** ignores the request to link a file in *lost+found*. This will always invoke the CLEAR error condition in Phase 4. Check size and contents of *lost+found*.

(CLEAR)

The inode mentioned in the immediately previous error condition can not be reconnected. See the part "Link Count" for more information.

Possible responses to CLEAR prompt are:

YES — Deallocate inode mentioned in the immediately previous error condition by zeroing its contents.

NO — Ignore this error condition.

LINK COUNT FILE I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST)

The link count for inode *I*, which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed. See the part "Link Count" for more information.

Possible responses to ADJUST prompt are:

YES — Replace link count of file inode *I* with *Y*.

NO — Ignore this error condition.

LINK COUNT DIR I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST)

The link count for inode *I*, which is a directory, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed.

Possible responses to ADJUST prompt are:

YES Replace link count of directory inode *I* with *Y*.

NO Ignore this error condition.

LINK COUNT F I=I OWNER=O MODE=M SIZE=S MTIME=T COUNT=X SHOULD BE Y (ADJUST)

The link count for *F* inode *I* is *X* but should be *Y*. The file name *F*, owner *O*, mode *M*, size *S*, and modify time *T* are printed.

Possible responses to ADJUST prompt are:

YES Replace link count of inode *I* with *Y*.

NO Ignore this error condition.

UNREF FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)

Inode *I*, which is a file, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. (See the parts "Link Counts" and "Data Blocks".) If the **−n** option is not set and the file system is not mounted, empty files will be cleared automatically.

Possible responses to CLEAR prompt are:

YES Deallocate inode *I* by zeroing its contents.

NO Ignore this error condition.

UNREF DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)

Inode *I*, which is a directory, was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. If the **−n** option is not set and the file system is not mounted, empty directories will be cleared automatically. Nonempty directories will not be cleared.

Possible responses to CLEAR prompt are:

YES Deallocate inode *I* by zeroing its contents.

NO Ignore this error condition.

BAD/DUP FILE I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with file inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. See the parts "Duplicate Blocks" and "Bad Blocks" for more information.

Possible responses to CLEAR prompt are:

YES Deallocate inode *I* by zeroing its contents.

NO Ignore this error condition.

BAD/DUP DIR I=I OWNER=O MODE=M SIZE=S MTIME=T (CLEAR)

Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed.

Possible responses to CLEAR prompt are:

YES — Deallocate inode *I* by zeroing its contents.

NO — Ignore this error condition.

FREE INODE COUNT WRONG IN SUPERBLK (FIX)

The actual count of the free inodes does not match the count in the superblock of the file system. (See the part "Free-Inode Count".) If the **-q** option is specified, the count will be fixed automatically in the superblock.

Possible responses to FIX prompt are:

YES — Replace count in superblock by actual count.

NO — Ignore this error condition.

H. PHASE 5: CHECK FREE LIST

This phase concerns itself with the free-block list. This part lists error conditions resulting from bad blocks in the free-block list, bad free-blocks count, duplicate blocks in the free-block list, unused blocks from the file system not in the free-block list, and the total free-block count incorrect.

EXCESSIVE BAD BLKS IN FREE LIST (CONTINUE)

The free-block list contains more than a tolerable number (usually 10) of blocks with a value less than the first data block in the file system or greater than the last block in the file system. See the parts "Free-Block List" and "Bad Blocks" for more information.

Possible responses to CONTINUE prompt are:

YES — Ignore rest of the free-block list and continue execution of **fsck**. This error condition will always invoke "BAD BLKS IN FREE LIST" error condition in Phase 5.

NO — Terminate program.

EXCESSIVE DUP BLKS IN FREE LIST (CONTINUE)

The free-block list contains more than a tolerable number (usually 10) of blocks claimed by inodes or earlier parts of the free-block list.

Possible responses to CONTINUE prompt are:

YES — Ignore the rest of the free-block list and continue execution of **fsck**. This error condition will always invoke "DUP BLKS IN FREE LIST" error condition in Phase 5.

NO — Terminate program.

BAD FREEBLK COUNT

The count of free blocks in a free-list block is greater than 50 or less than 0. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5.

X BAD BLKS IN FREE LIST

X blocks in the free-block list have a block number lower than the first data block in the file system or greater than the last block in the file system. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5. See the parts "Free-Block List" and "Bad Blocks" for more information.

X DUP BLKS IN FREE LIST

X blocks claimed by inodes or earlier parts of the free-list block were found in the free-block list. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5.

X BLK(S) MISSING

X blocks unused by the file system were not found in the free-block list. This error condition will always invoke the "BAD FREE LIST" condition in Phase 5. See the part "Free-Block List" for more information.

FREE BLK COUNT WRONG IN SUPERBLOCK (FIX)

The actual count of free blocks does not match the count in the superblock of the file system. See the part "Free-Block Count" for more information.

Possible responses to FIX prompt are:

YES	Replace count in superblock by actual count.
NO	Ignore this error condition.

BAD FREE LIST (SALVAGE)

Phase 5 has found bad blocks in the free-block list, duplicate blocks in the free-block list, or blocks missing from the file system. If the **-q** option is specified, the free-block list will be salvaged automatically.

Possible responses to SALVAGE prompt are:

YES	Replace actual free-block list with a new free-block list. The new free-block list will be ordered to reduce time spent by the disk waiting for the disk to rotate into position.
NO	Ignore this error condition.

I. PHASE 6: SALVAGE FREE LIST

This phase concerns itself with the free-block list reconstruction. This part lists error conditions resulting from the blocks-to-skip and blocks-per-cylinder values.

Default free-block list spacing assumed

This is an advisory message indicating the blocks-to-skip is greater than the blocks-per-cylinder, the blocks-to-skip is less than one, the blocks-per-cylinder is less than one, or the blocks-per-cylinder is greater than 1000.

The default values of 7 blocks-to-skip and 400 blocks-per-cylinder are used. See **fsck**(1M) in the UNIX System Administrator's Manual for further details.

J. CLEANUP

Once a file system has been checked, a few cleanup functions are performed. This part lists advisory messages about the file system and modify status of the file system.

X files Y blocks Z free

This is an advisory message indicating that the file system checked contained *X* files using *Y* blocks leaving *Z* blocks free in the file system.

******* BOOT UNIX (NO SYNC!) *******

This is an advisory message indicating that a mounted file system or the root file system has been modified by **fsck**. If the UNIX system is not rebooted immediately, the work done by **fsck** may be undone by the in-core copies of tables the UNIX system keeps.

******* FILE SYSTEM WAS MODIFIED *******

This is an advisory message indicating that the current file system was modified by **fsck**. If this file system is mounted or is the current root file system, **fsck** should be halted and the UNIX system rebooted. If the UNIX system is not rebooted immediately, the work done by **fsck** may be undone by the in-core copies of tables.

NOTES